Grundlagen der Informatik III

Wintersemester 2013/2014

Prof. Dr.-Ing. Michael Goesele,
Simon Fuhrmann, Fabian Langguth

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Praktikum

23.10.2013

Das Praktikum wird in Teams von drei Personen bearbeitet, die alle der selben Übungsgruppe angehören müssen. Vermerken Sie im Quellcode alle Teammitglieder mit Matrikelnummern. Die Lösung muss bis **spätestens 06.11.** im Moodle hochgeladen werden. Es genügt, wenn *ein* Teammitglied die Lösung abgibt. Den eigentlichen Testattermin vereinbaren Sie selbst mit Ihrem Tutor.

Geben Sie alle zum Kompilieren und Ausführen Ihres Programms benötigten Dateien in einem .zip- oder .tar.gz-Archiv ab. Ihre Abgabe muss mit den auf dem Aufgabenblatt angegebenen Anweisungen auf den RBG-Poolrechnern kompilier- und ausführbar sein, ansonsten wird sie mit 0 Punkten bewertet.

Wenn Sie dieses Praktikum erfolgreich bearbeiten und die Klausur bestehen, erhalten Sie **einen zusätzlichen Klausurpunkt**.

ACHTUNG: Aufgrund von Änderungen and den RBG Systemen werden die Zugriffe auf RBG Rechner vom 1.11.2013 - 18 Uhr bis zum 04.11.2013 - 8 Uhr gesperrt. Weitere Informationen finden Sie hier: `http://bit.ly/1bQcT8A`.

Vorbemerkungen

In diesem Praktikum arbeiten Sie mit einer Rastergrafik. Dazu finden Sie in der Datei `globals.S` drei Variablen:

- An der Speicherstelle `width` steht die Breite des Bildes in Pixeln als 32-bit Ganzzahl
- An der Speicherstelle `height` steht die Höhe des Bildes in Pixeln als 32-bit Ganzzahl
- Ab der Speicherstelle `buf` stehen `width` * `height` Bytes mit Pixelwerten von 0 für Schwarz bis 255 für Weiß. Das Pixel an der Position (x, y) steht an der Speicherstelle `buf + (y * width) + x`.

Sie können das Pixel (20, 10) demnach wie folgt auf Weiß setzen:

```
1  movl $10, %eax
2  imull width, %eax
3  addl $20, %eax
4  movb $255, buf(%eax)
```

Gehen Sie bei der Implementation davon aus, dass die zu zeichnenden Figuren vollständig innerhalb des Bildes liegen. Führen Sie daher *keine* Bereichsprüfung durch.

Versuchen Sie, bei der Erstellung des Assemblercodes strukturiert vorzugehen. Planen Sie zunächst die Belegung der Register. Wenn Sie mehr Variablen benötigen als Sie Register haben, deklarieren Sie globale Variablen im Abschnitt `.data`. Vergessen Sie nicht, Register für Zwischenergebnisse einzuplanen.

Gehen Sie dann den Algorithmus Zeile für Zeile durch und überlegen Sie, wie Sie die jeweilige Operation in Assembler ausführen können. Funktionen wie `abs(x)` zur Bestimmung des Absolutwerts einer Zahl müssen Sie dabei in Teilschritte zerlegen (in diesem Fall mit einer Fallunterscheidung ob $x < 0$).

Ihre Lösung muss mit diesen Befehlen auf den RBG-Poolcomputern kompilier- und lauffähig sein:

```
make thales
./thales | display -
```

Wenn Sie sich per SSH auf die RBG-Rechner verbinden, aktivieren Sie X11-Forwarding mit der Option `-X`.

Alternativ können Sie natürlich das Ausgabebild in eine Datei speichern und diese mit einem Bildbetrachter Ihrer Wahl ansehen.

```
make thales
./thales >thales.pgm
```

Zur Fehlersuche kann es hilfreich sein, das Programm in einem Debugger zu starten. Auf den RBG-Poolrechnern ist dazu der GNU Debugger `gdb` installiert. Starten Sie diesen mit `gdb thales`, nachdem Sie das Programm wie oben mit `make thales` kompiliert haben. Hier ein Beispiel zur Verwendung einiger typischer Befehle:

```
(gdb) break draw_line
Breakpoint 1 at 0x8048678: file draw_line.S, line 9.
(gdb) run
Starting program: .../pr1/thales
Breakpoint 1, draw_line () at draw_line.S:9
10    pusha
(gdb) info registers
eax            0xffffdc34  -9164
ecx            0xc650d92b  -967780053
edx            0x1   1
ebx            0xf7fb7ff4  -134512652
esp            0xffffdb6c  0xffffdb6c
ebp            0xffffdb88  0xffffdb88
esi            0x0   0
edi            0x0   0
eip            0x8048678   0x8048678 <draw_line>
eflags         0x282 [ SF IF ]
cs             0x23  35
ss             0x2b  43
ds             0x2b  43
es             0x2b  43
fs             0x0   0
gs             0x63  99
(gdb) cont
Continuing.

Breakpoint 1, draw_line () at draw_line.S:9
10    pusha
(gdb) print $ebp
$1 = (void *) 0xffffdb88
(gdb) quit
```

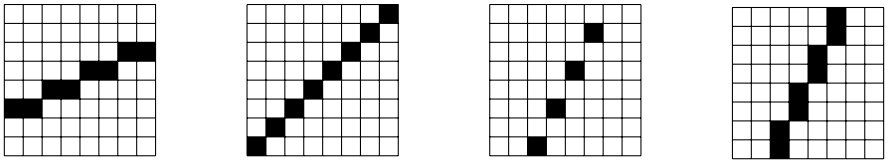
Weiterhin können Sie mit `call dump()` im Debugger den aktuellen Bildinhalt auf die Konsole ausgeben lassen, dabei werden alle Pixel, die nicht Schwarz sind, mit `X` markiert.

Aufgabe: Der Bresenham-Algorithmus

Es gibt verschiedene Möglichkeiten, eine Linie in eine Rastergrafik einzuzeichnen. Ein einfacher Ansatz, um eine Linie zwischen den Punkten (x0, y0) und (x1, y1) zu zeichnen wäre zum Beispiel:

```
float m = (float)(y1 - y0) / (float)(x1 - x0);
for (int x = x0; x <= x1; ++x)
{
   int y = m * (x - x0) + y0;
   setPixel(x, y);
}
```

Dieser Ansatz liefert jedoch unschöne Ergebnisse für $|m| > 1$, sodass mindestens eine Fallunterscheidung nötig wird, die für $|y1 - y0| > |x1 - x0|$ die Schleife über y laufen lässt und x anhand der Steigung berechnet.



$|m| < 1$ $|m| = 1$ $|m| > 1$ angepasst

Obwohl dieser Ansatz simpel wirkt, ist er aus Rechnerperspektive aufwendig: In jedem Schleifendurchlauf müssen Ganzzahlen in Fließkommazahlen und wieder zurück umgewandelt werden. Ein Algorithmus, der auf Jack Bresenham zurückgeht, kommt hingegen mit Ganzzahl-Additionen und -Subtraktionen aus (die Multiplikation in Zeile 27 kann durch einen Shift implementiert werden).

```
1  void draw_line(int x0, int y0, int x1, int y1)
2  {
3     int dx = abs(x1 - x0);
4     int sx;
5     if (x0 < x1)
6        sx = 1;
7     else
8        sx = -1;
9
10    int dy = -abs(y1 - y0); /* negativ! */
11    int sy;
12    if (y0 < y1)
13       sy = 1;
14    else
15       sy = -1;
16
17    int err = dx + dy;
18
19    int x = x0;
20    int y = y0;
21
22    while (1) {
23       setPixel(x, y);
24       if (x == x1 && y == y1)
25          break;
26
27       int e2 = 2 * err;
28       if (e2 > dy) {
29          err = err + dy;
30          x = x + sx;
31       }
32       if (e2 < dx) {
33          err = err + dx;
34          y = y + sy;
35       }
36    }
37 }
```

(vgl. `http://de.wikipedia.org/wiki/Bresenham-Algorithmus#Kompakte_Variante`)

Sie sollen nun diesen Algorithmus in Assembler implementieren. Setzen Sie die Pixel dabei auf Weiß (255). In der Vorlage finden Sie ein Beispielprogramm `thales.c`, das mit ihrer Implementation drei Linien in ein kleines Bild einzeichnen wird. Fügen Sie Ihre Implementation in die Datei `draw_line.S` ein.

Bewertung

Sie erhalten *einen Punkt* auf die korrekte Implementation von `draw_line`.

Sie erhalten *keine* Punkte, wenn Ihr Programm nicht ohne Veränderung auf den RBG-Poolrechnern kompilierbar und lauffähig ist. Testen Sie Ihr Programm daher *unbedingt* vor der Abgabe. Achten Sie außerdem – auch in Ihrem eigenen Interesse – auf lesbaren, gut kommentierten Code. Sie werden im Testat Code-Ausschnitte erklären müssen.